Adaptive Software for a Changing World

Assessing Robustness Properties in Dynamic Discovery of Ad Hoc Network Services

Christopher Dabrowski and Kevin Mills

Briefing for Sun Microsystems
Tech-a-Tech
Burlington, MA
October 4, 2001

Presentation Roadmap

- Project Objectives, Motivation, and Goals
- Modeling & Analysis
  - Architecture-based approach
  - Generic UML structural model
  - Specific models instantiated with Architecture Description Language
- Previous Work
  - Verifying our approach - using Jini as an example
- Overview of On-Going Work
  - How do different service discovery architectures respond to node and link failures?
  - How can these responses be improved?
- Plans for Future Work

Dynamic discovery protocols..

enable **network elements** (including software clients and services, and devices):

(1) to **discover** each other without prior arrangement,
(2) to **express** opportunities for collaboration,
(3) to **compose** themselves into larger collections that cooperate to meet an application need, and
(4) to **detect and adapt to changes** in network topology.

NIST/ITL role in supporting industry …

- In the future, all **software systems will be distributed** systems written to operate over a network, where conditions vary.
- Dynamic **discovery protocols provide a foundation** upon which such distributed systems will be constructed.
- **Understanding** the current (first) generation of discovery protocols is **essential** to enable industry to improve designs for the second and subsequent generations.
- Our project applies architecture-based analysis, languages, and tools **to help industry improve designs and specifications for service discovery protocols *and*** Architectural Description Languages (ADLs) and tools.

DoD Programs Related to this Project

- Fault Tolerant Networks – DARPA Program
  http://www.darpa.mil/ito/research/ftn/index.html

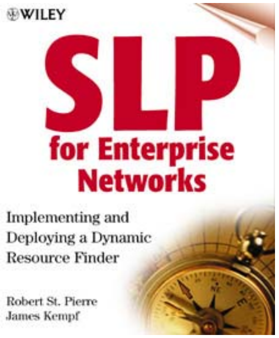
- OpenWings – Joint Motorola-Sun-U.S. Army Program (key enabler of Joint Vision 2020)
  http://www.openwings.org/index.htm

- Organically Assured and Survivable Information Systems (OASIS) – DARPA Program
  http://www.darpa.mil/ato/programs/oasis.htm

- Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA) - DARPA Program
  http://www.darpa.mil/ito/research/dasada/index.html

- Critical Infrastructure Protection (CIP) and High Confidence, Adaptable Software (SW) Research Program of the University Research Initiative (URI) – Office of Naval Research
  http://www.onr.navy.mil/sci_tech/special/cipswuri/

Selected Current (First) Generation Protocols for Dynamic Service Discovery


WILEY

SLP for Enterprise Networks

Implementing and Deploying a Dynamic Resource Finder

Robert St. Pierre
James Kempf

Our Goal

To provide **metrics** and approaches to **compare and contrast emerging** dynamic **discovery protocols**, **to better understand** their critical functions, to identify weaknesses, and to strengthen the robustness, quality and correctness of designs for future protocols.

Our Overall Technical Approach

- Build a **generic, domain model** (UML) providing consistent terminology encompassing a range of service discovery protocols.
- Build **executable models** of service discovery protocols from extant specifications, and analyze them under conditions of dynamic change.
- Build **measurement infrastructure** and measure implementations of dynamic service protocols for scalability.
- Build **simulation models** of service protocols and assess the performance of such models in the face of dynamic change.
- Design, model, and evaluate **protocol mechanisms** that enable discovery protocols to self-adapt in the face of dynamic change *(this part of the project is funded by the DARPA Fault Tolerant Networks program).*

Presentation Roadmap

- Project Objectives, Motivation, and Goals
- Modeling & Analysis
  - Architecture-based approach
  - Generic UML structural model
  - Specific models instantiated with Architecture Description Language
- Previous Work
  - Verifying our approach - using Jini as an example
- Overview of On-Going Work
  - How do different service discovery architectures respond to node and link failures?
  - How can these responses be improved?
- Plans for Future Work

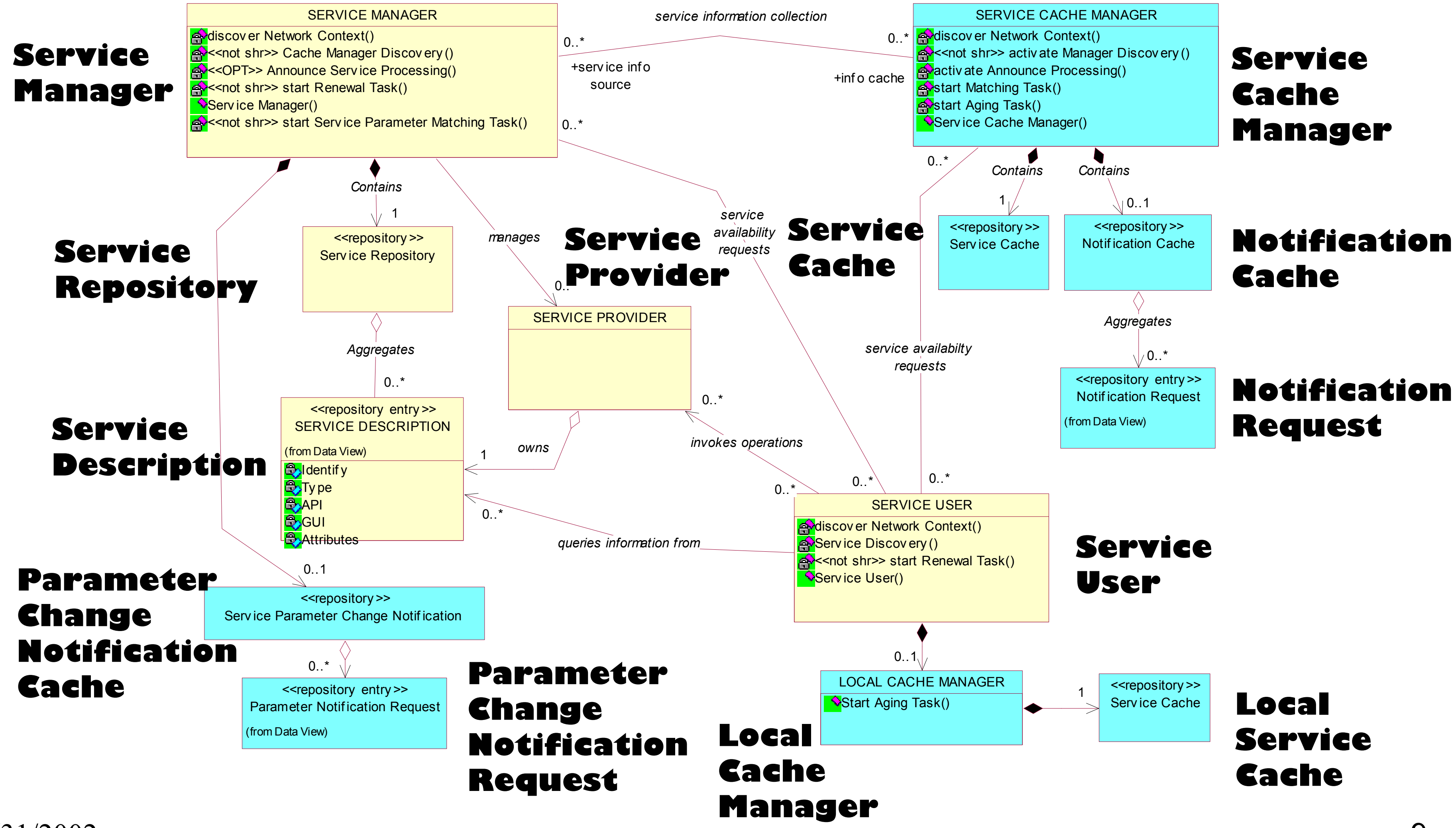
Technical Approach Specific to Architectural Modeling & Analysis

- **Model** Discovery Protocol **specifications using Architectural Description Languages** (ADLs) and associated tools

- **Analyze** Discovery Protocol **models** to assess consistency, correctness, and completeness under conditions of dynamic change.

- **Compare and contrast** our **models** with regard to function, structure, behavior, performance, complexity, and scalability under conditions of dynamic change.

Foundation for Comparisons: A Generic Structural Model (UML) for Service-Discovery Domain



Service Manager

SERVICE MANAGER
- discover Network Context()
- <<not shr>> Cache Manager Discovery()
- <<OPT>> Announce Service Processing()
- <<not shr>> start Renewal Task()
- Service Manager()
- <<not shr>> start Service Parameter Matching Task()

service information collection
0..* +service info source
0..* +info cache
0..*

Service Cache Manager

SERVICE CACHE MANAGER
- discover Network Context()
- <<not shr>> activate Manager Discovery()
- activate Announce Processing()
- start Matching Task()
- start Aging Task()
- Service Cache Manager()

Contains 1
Contains 0..1
0..*

Service Repository

<<repository>> Service Repository

Contains 1
Aggregates 0..*

manages 0..

Service Provider

SERVICE PROVIDER

service availability requests

Service Cache

<<repository>> Service Cache

Notification Cache

<<repository>> Notification Cache

Aggregates 0..*

Notification Request

<<repository entry>> Notification Request (from Data View)

service availabilty requests

Service Description

<<repository entry>> SERVICE DESCRIPTION (from Data View)
- Identify
- Type
- API
- GUI
- Attributes

owns 1
0..* invokes operations 0..*
0..* 0..*
0..* queries information from

Service User

SERVICE USER
- discover Network Context()
- Service Discovery()
- <<not shr>> start Renewal Task()
- Service User()

0..1

Parameter Change Notification Cache

<<repository>> Service Parameter Change Notification

0..*

<<repository entry>> Parameter Notification Request (from Data View)

Parameter Change Notification Request

0..1

Local Cache Manager

LOCAL CACHE MANAGER
- Start Aging Task()

1

<<repository>> Service Cache

Local Service Cache

Architectual Description Languages & Tools….

- **Represent essential complexity** of service discovery protocols with effective abstractions
  - *Rapide*, public-domain ADL and toolset developed at Stanford University for DARPA, provides ability to execute architecture specifications, producing Partially Ordered Sets of Events (POSETs) for analysis.
- Provide a framework and context
  - to **compare and contrast** dynamic **service discovery architectures**
  - to **define metrics** that yield qualitative and quantitative measures of dynamic component-based software
  - to **model alternate approaches** to specific functions or mechanisms where permitted by a specification or where a specification appears ambiguous
  - to help **pinpoint** where **inconsistencies and ambiguities** may exist within software implementing specifications & to understand how such issues arise
- Provide our work to ADL purveyors and researchers for use in improving future languages and tools

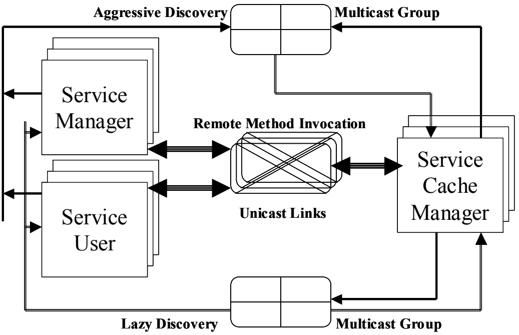
Architecture-based Approach to Modeling and Analysis

(using Rapide, an Architecture Description Language and Tools Developed for DARPA by Stanford)

Time	Command	Parameters
5	NodeFail	SM4
5	LinkFail	SCM1 SM4
10	GroupJoin	SM4 GROUP1
10	FindService	SU8 5 1 2 S XYZ ALL
50	AddService	SM4 SCM3 T ATT API GUI 20 30

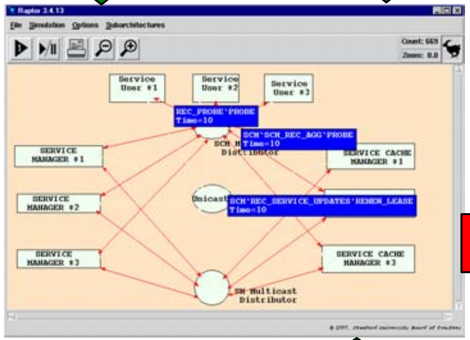
Scenario

Topology

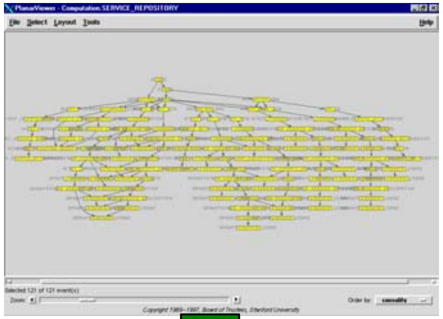


Specification Model

```
-- ********************************************************
-- ** 3.3  DIRECTED DISCOVERY CLIENT INTERFACE      **
-- ********************************************************
-- This is used by all JINI entities in directed
-- discovery mode.  It is part of the SCM_Discovery
-- Module. Sends Unicast messages to  SCMs on list of
-- SCMS to be discovered until all SCMS are found.
-- Receives updates from SCM DB of discovered SCMs and
-- removes SCMs accordingly
-- NOTE: Failure and recovery behavior are not
-- yet defined and need reviw.
TYPE Directed_Discovery_Client
  (SourceID : IP_Address; InSCMsToDiscover : SCMList; StartOption : DD_Code;
   InRequestInterval : TimeUnit; InMaxNumTries : integer; InPV : ProtocolVersion)
IS INTERFACE
SERVICE DDC_SEND_DIR    : DIRECTED_2_STEP_PROTOCOL;
SERVICE DISC_MODES      : dual SCM_DISCOVERY_MODES;
SERVICE DD_SCM_Update   : DD_SCM_Update;
SERVICE SCM_Update      : SCM_Update;
SERVICE DB_Update       : dual DB_Update;
SERVICE NODE_FAILURES : NODE_FAILURES;  -- events for failure and recovery.
ACTION
 IN Send_Requests(),
    BeginDirectedDiscovery();
BEHAVIOR
   action animation_Iam (name: string);
   MySourceID         : VAR IP_Address;
   PV                 : VAR ProtocolVersion;
```



Execute with Rapide



Consistency Conditions

For All (SM, SD, SCM):
(SM, SD) IsElementOf SCM registered-services (CC1)
implies SCM IsElementOf SM discovered-SCMs

For All (SM, SD, SCM):
SCM IsElementOf SM discovered-SCMs & (CC2)
(SD) IsElementOf SM managed-services
implies (SM, SD) IsElementOf SCM registered-services

For All (SM, SD, SCM):
SCM IsElementOf SM discovered-SCMs & (CC3)
(SM, SD) IsElementOf SCM registered-services &
NOT (SCM IsElementOf SM persistent-list)
implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)

For All (SM, SD, SCM, SU, NR):
(SU, NR) IsElementOf SCM requested-notifications & (CC4)
(SM, SD) IsElementOf SCM registered-services &
Matches((SM, SD), (SU,NR))
implies (SM, SD) IsElementOf SU matched-services

Analyze POSETs

Assess Correctness, Performance, & Complexity

Presentation Roadmap

- Project Objectives, Motivation, and Goals

- Modeling & Analysis
  - Architecture-based approach
  - Generic UML structural model
  - Specific models instantiated with Architecture Description Language

- Previous Work
  - Verifying our approach - using Jini as an example

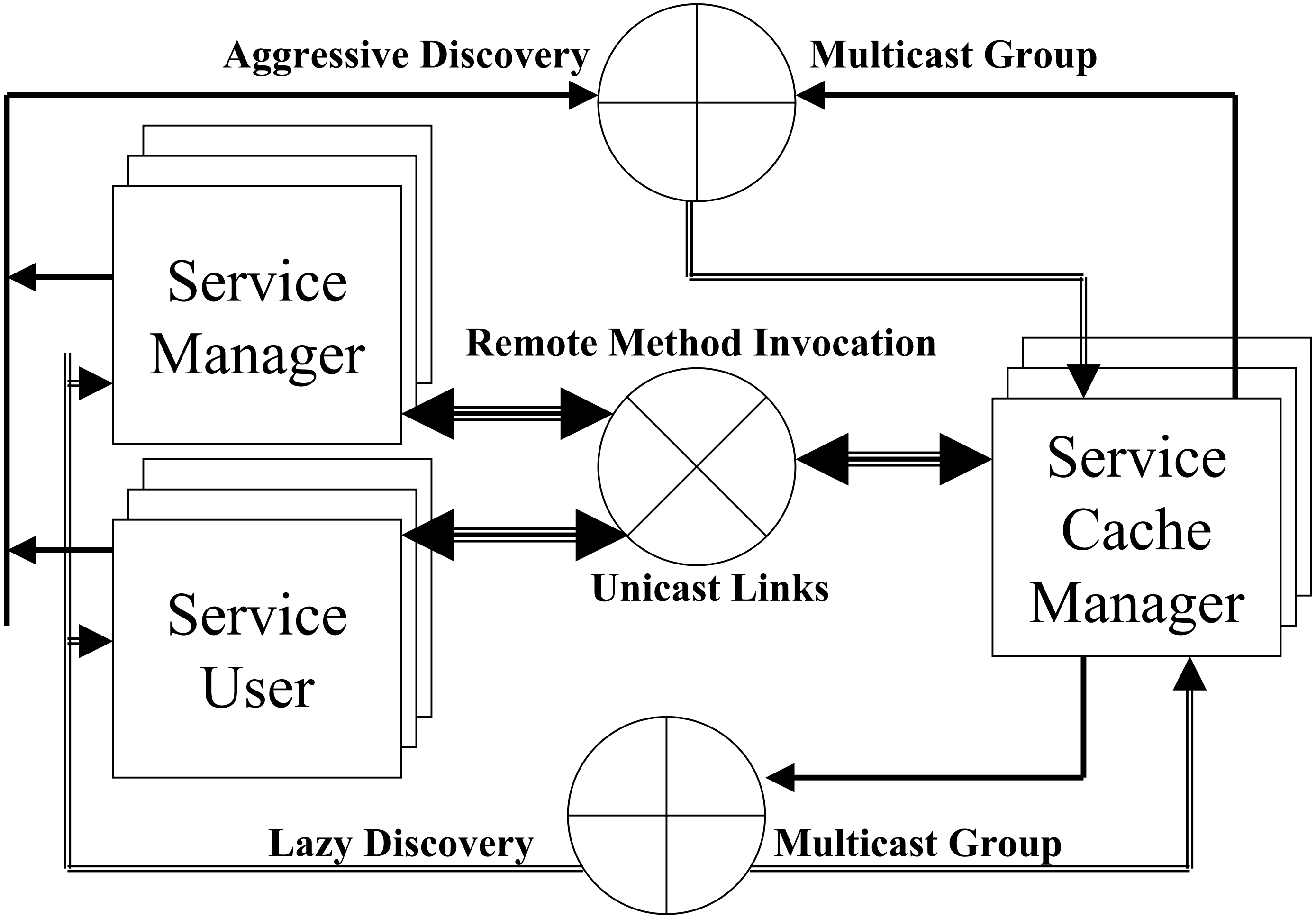
- Overview of On-Going Work
  - How do different service discovery architectures respond to node and link failures?
  - How can these responses be improved?

- Plans for Future Work

Analysis of Jini Using Architecture-Based Approach

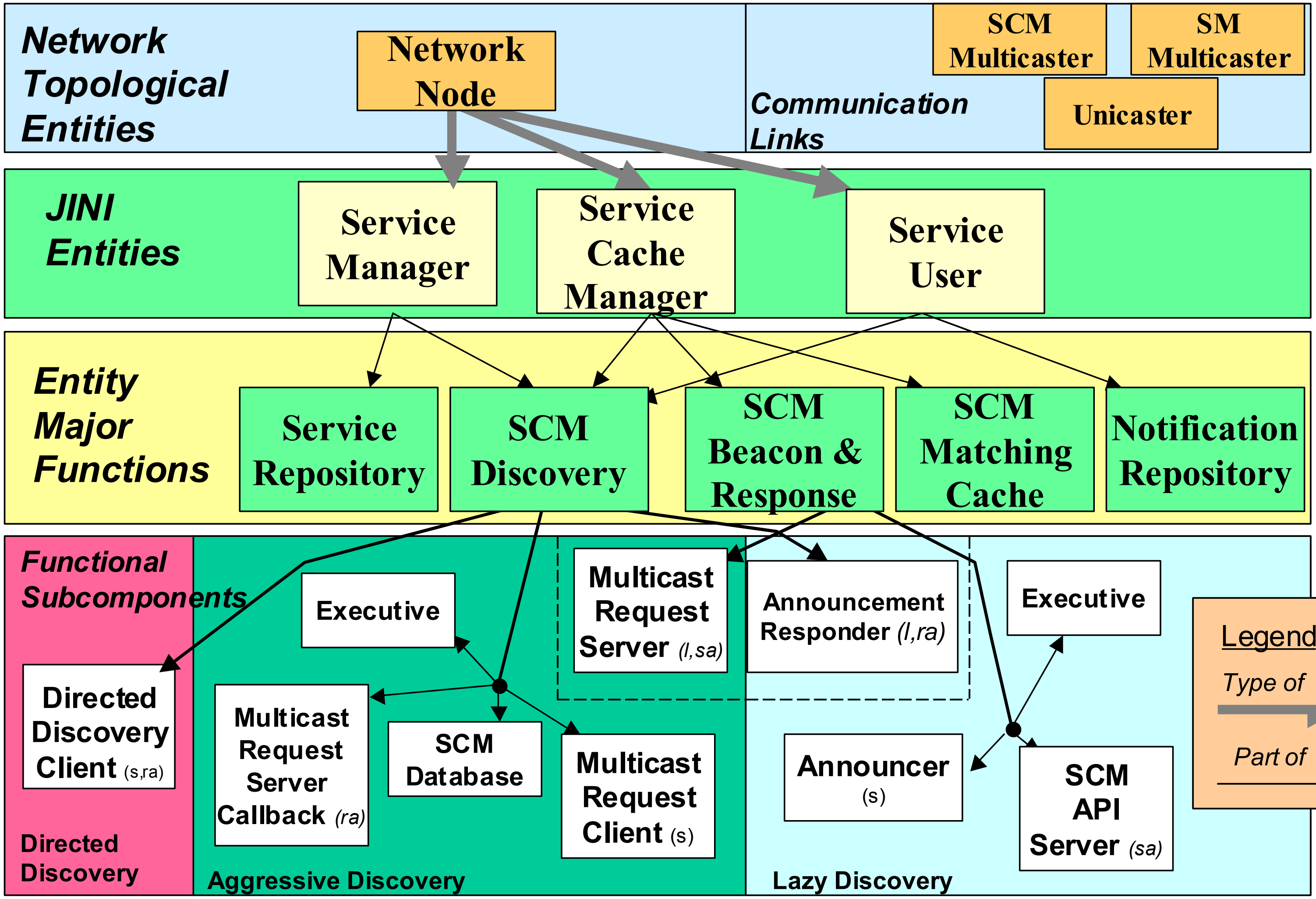
- **Architecture** depicts network topological entities, Jini entities and major functions, and key behavior
- **Consistency conditions** posit state relationships a protocol should strive to maintain among functional entities
- **Scenarios** trigger possible sequences of events

Sample Network Topology Applicable to Jini Entities

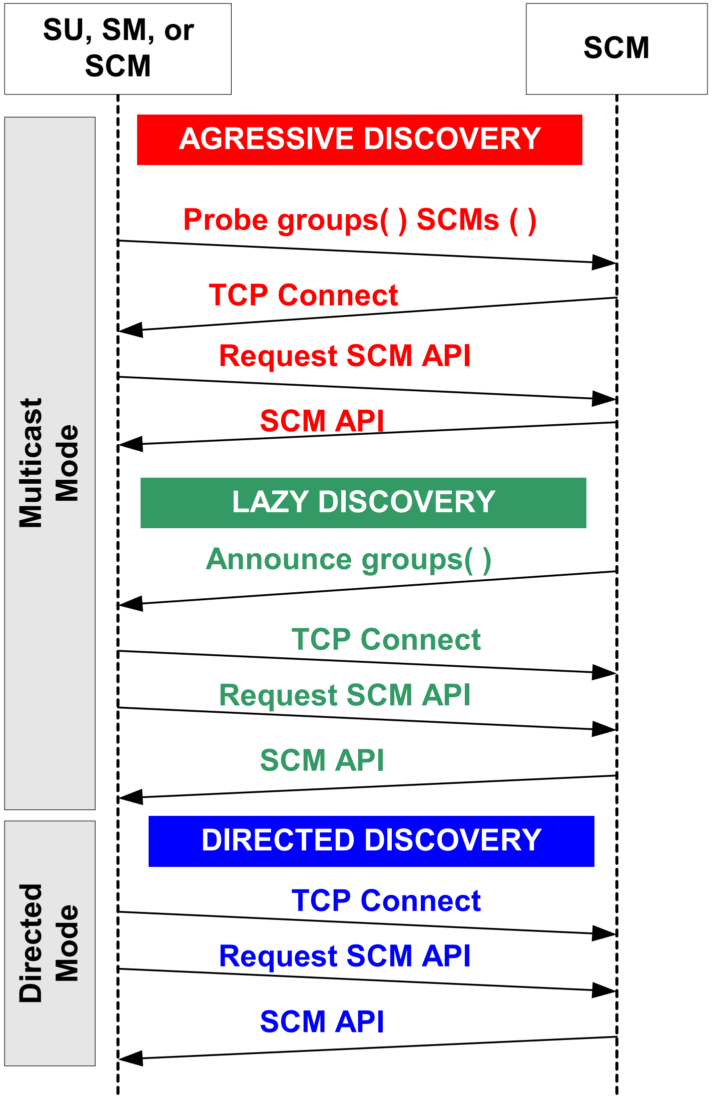


Layered View of Prototype JINI Architecture in Rapide

Derived from SEI Architectural Layers Approach



Representing Jini Discovery in Terms of Our Model



- In Multicast Mode, an SU, SM, or SCM will try to discover SCMs that are members of the same *group* as the discovering entity.

  And an SU, SM, or SCM <u>may dynamically join or leave *groups*</u>

- In Directed Mode, an SU, SM, or SCM will try to discover SCMs that are on a *persistent list* of SCMs to be discovered maintained by the discovering entity.

  And SCMs to be discovered may be <u>dynamically added or removed from the *persistent list*</u>.

Real-Time Checking of Consistency Conditions

Sample Consistency Condition #4 (race condition)

For All (SM, SD, SCM, SU, NR):
  (SU, NR) *IsElementOf* SCM requested-notifications &
  (SM, SD) *IsElementOf* SCM registered-services &
  Matches ((SM, SD), (SU, NR))
    implies (SM, SD) *IsElementOf* (SU matched-services)

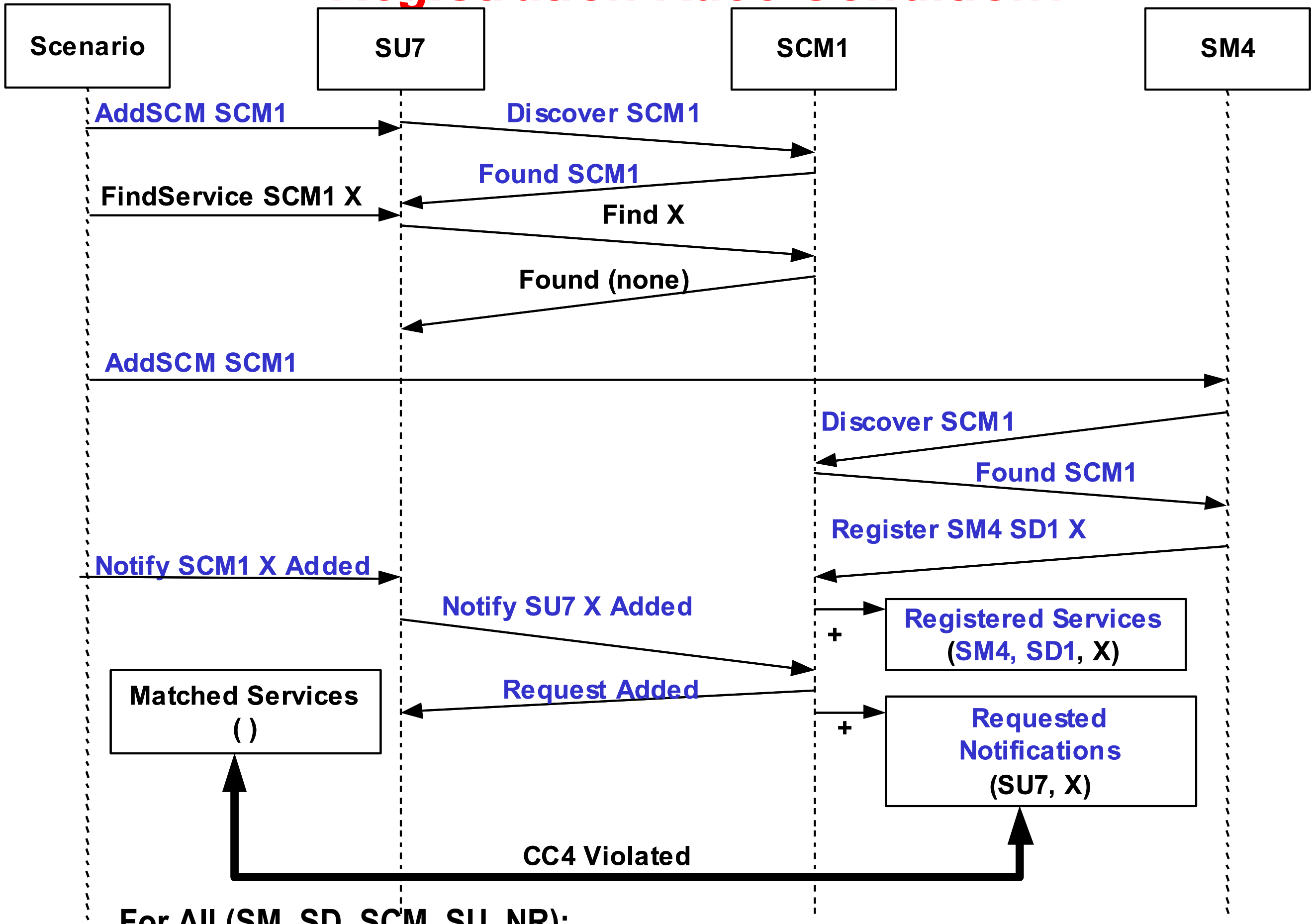
…that is, if an SU has requested notification with a Service Cache Manager of a service that matches a service description registered by a Service Manager on the same Cache Manager, then that service description should be provided to the Service User.

*Assuming absence of network failure and normal delays due to updates

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager
- SU is Service User
- NR is Notification Request

- requested-notifications is a set of (SU,NR) pairs maintained by the SCM
- registered-services is a set of (SM,SD) pairs maintained by the SCM
- matched-services is the set of (SM,SD) pairs maintained by the SU

Should the Jini Specification Advise about Possibility for Registration Race Condition?



For All (SM, SD, SCM, SU, NR):
(SU, NR) IsElementOf SCM requested-notifications & (CC4)
(SM, SD) IsElementOf SCM registered-services &
Matches((SM, SD), (SU,NR))
implies (SM, SD) IsElementOf SU matched-services

Real-Time Checking of Consistency Conditions

Sample Consistency Condition #3

For All (SM, SD, SCM):
SCM *IsElementOf* SM discovered-SCMs
(SM, SD) *IsElementOf* SCM registered-services
NOT (SCM IsElementOf SM persistent-list)
implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)

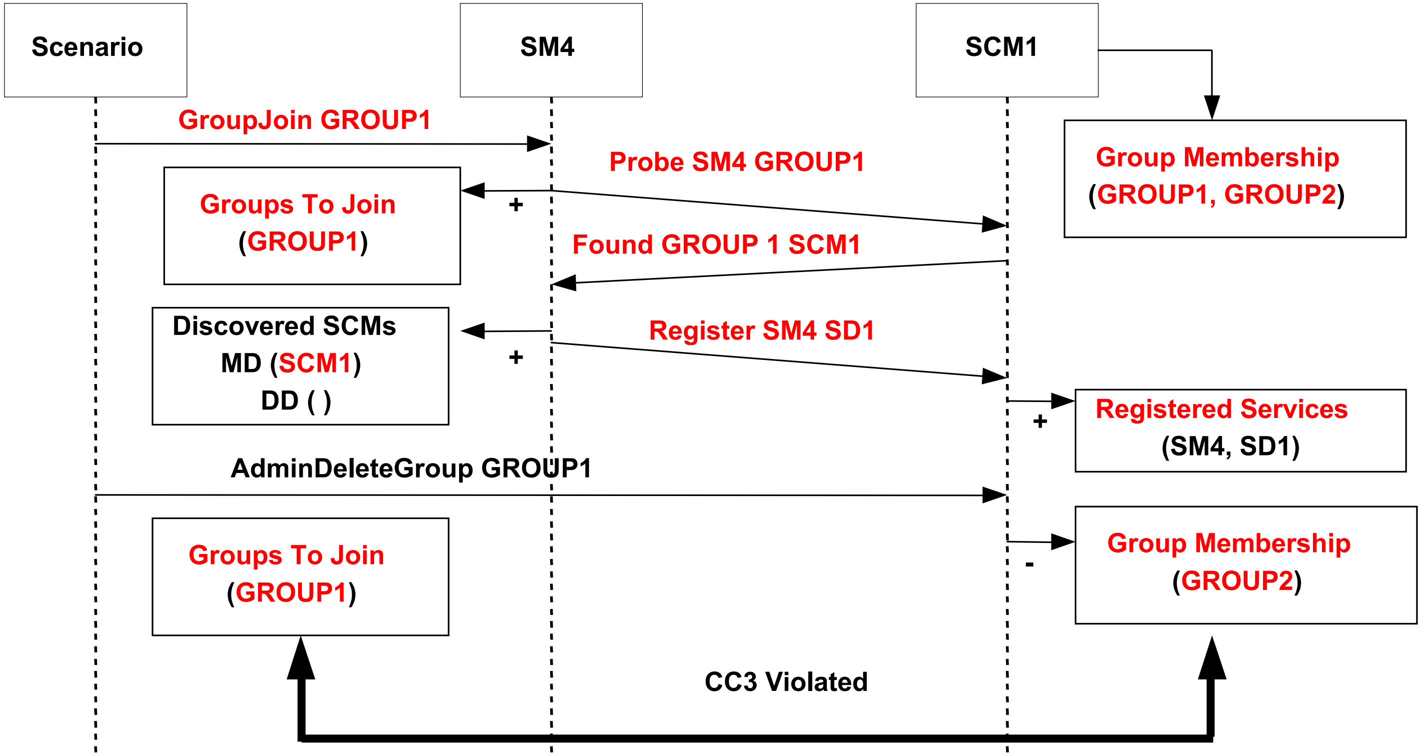
…that is, if a Service Manager has discovered, and registered its service descriptions on, a Service Cache Manager that is not on the Service Manager's persistent list, then the Service Manager must be seeking group membership in at least one group the Service Cache Manager belongs to.

*Assuming absence of network failure and normal delays due to updates

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager

- registered-services is a set of (SM,SD) pairs maintained by the SCM
- discovered-SCMs is a set of SCMs discovered by the SM
- Persistent-list is the set of SCMs the SM is seeking though directed discovery

What Might Happen When SCM Changes Group Membership Dynamically?



For All (SM, SD, SCM):

SCM IsElementOf SM discovered-SCMs & (CC3)
(SM, SD) IsElementOf SCM registered-services &
NOT (SCM IsElementOf SM persistent-list)
implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)

Real-Time Checking of Consistency Conditions

Sample Consistency Condition #1

For All (SM, SD, SCM): (SM, SD) *IsElementOf* SCM registered-services
implies SCM *IsElementOf* SM discovered-SCMs

…that is, a Service Manager should register its Services on an Service Cache Manager *only if* it maintains that Cache Manager on its “known SCM” LIst.

*Assuming absence of network failure and normal delays due to updates

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager

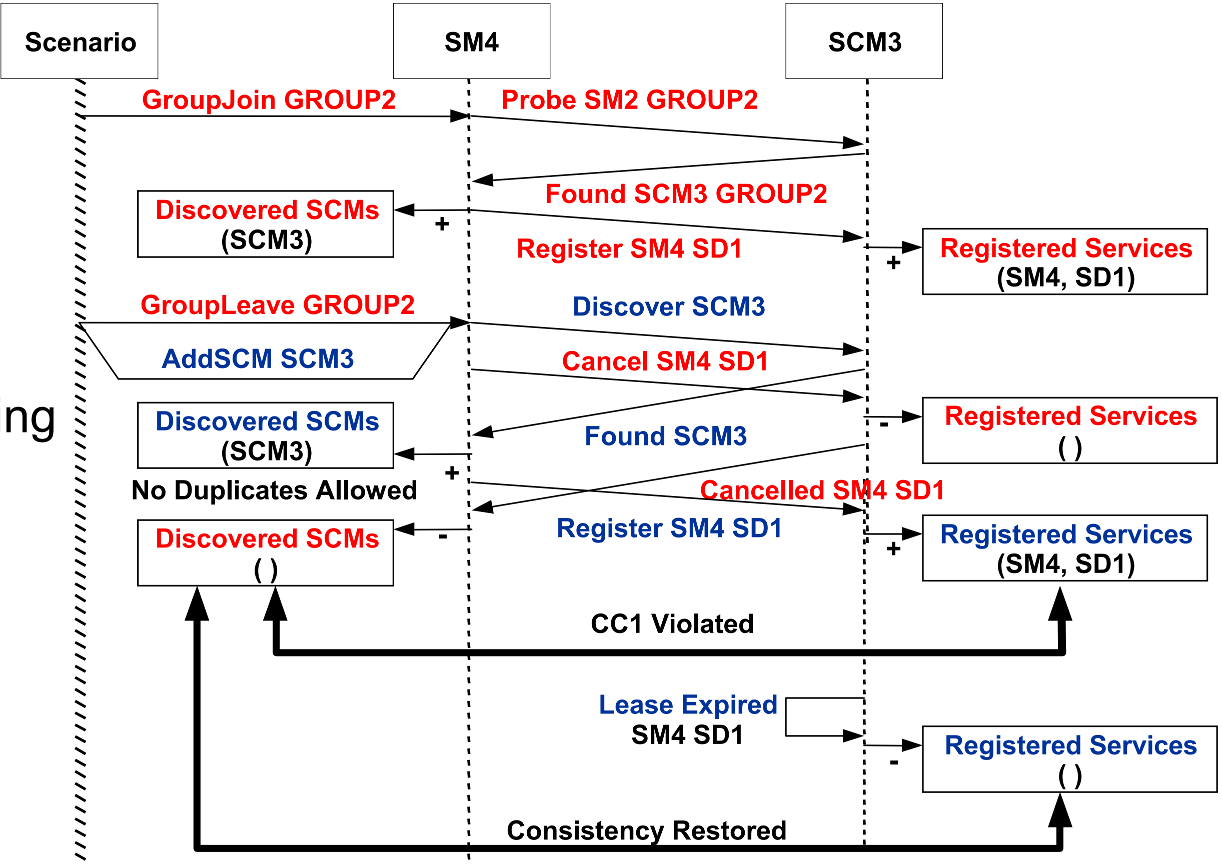
- registered-services is a set of (SM,SD) pairs maintained by the SCM
- discovered-SCMs is a set of SCMs discovered by the SM

Same executable model can be used to assess selected performance properties and to measure complexity+

+future work on the project intends to investigate the relationship between design complexity (applying ideas from Kolmogorov Complexity) and design quality (as represented by violation of consistency conditions)

Could the Jini Specification Lead to Implementations Exhibiting Undesired Interaction between Directed and Multicast Discovery?

Based on one possible interpretation of specification using a ***single-list assumption***



For All (SM, SD, SCM):

(SM, SD) IsElementOf SCM registered-services (CC1)
implies SCM IsElementOf SM discovered-SCMs

Real-Time Checking of Consistency Conditions (con't)

Sample Consistency Condition #2

For All (SM, SD, SCM):
SCM *IsElementOf* SM discovered-SCMs &
SD *IsElementOf* SM managed-services
implies (SM, SD) *IsElementOf* SCM registered-services

…that is, a Service Manager should register its Services on an Service Cache Manager *if* the Service Manager has discovered the Cache Manager and is maintaining the SCM identifier on its "known SCM" LIst.

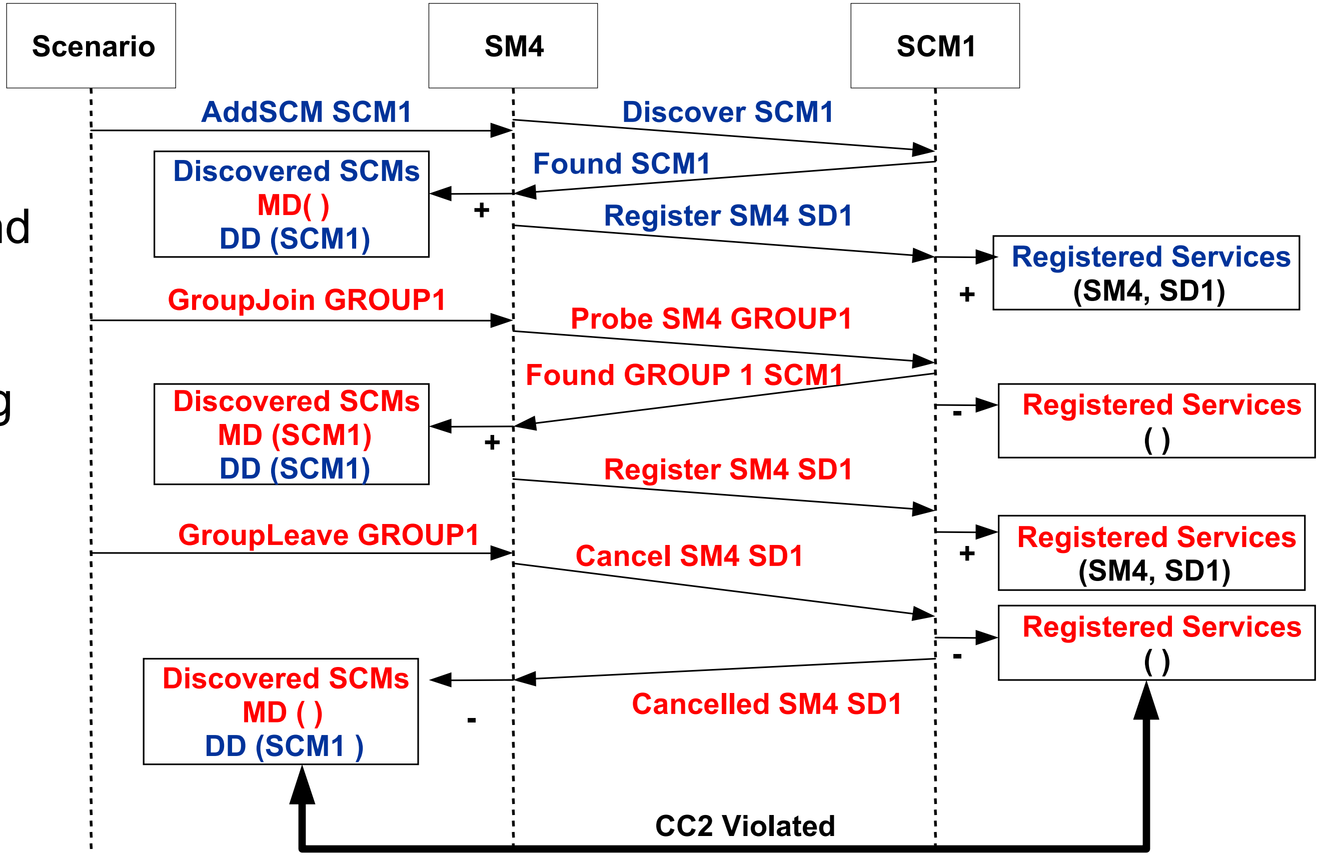
*Assuming absence of network failure and normal delays due to updates

- SM is Service Manager
- SD is Service Description
- SCM is Service Cache Manager

- discovered-SCMs is a set of SCMs discovered by the SM
- managed-services is a set of (SM,SD) pairs maintained by the SM
- registered-services is the set of (SM, SD) pairs maintained by the SCM

Could the Jini Specification Lead to Implementations Exhibiting Undesired Interaction between Directed and Multicast Discovery?

Based on a second possible interpretation of specification using ***separate-list assumption***



For All (SM, SD, SCM):

SCM IsElementOf SM discovered-SCMs & (CC2)
(SD) IsElementOf SM managed-services
implies (SM, SD) IsElementOf SCM registered-services

Presentation Roadmap

- Project Objectives, Motivation, and Goals
- Modeling & Analysis
  - Architecture-based approach
  - Generic UML structural model
  - Specific models instantiated with Architecture Description Language
- Previous Work
  - Verifying our approach - using Jini as an example
- Overview of On-Going Work
  - How do different service discovery architectures respond to node and link failures?
  - How can these responses be improved?
- Plans for Future Work

How do Two- and Three-Party Architectures for Service Discovery Respond to Failures?

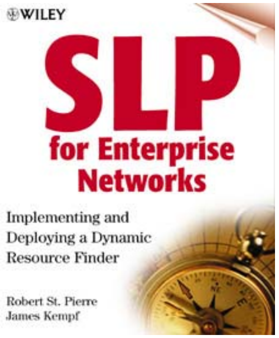
- **Two-Party vs. Three-Party architectures**
  Two alternative architectural designs that underlie commercial service discovery protocols, including Jini, UPnP, and Service Location Protocol

- **Impact of Study:**
  1. Provide valuable information to designers and users of service discovery protocols for **improving specifications**, thus **promoting software quality and reliability**.
  2. Create **generic set of test scenarios and related metrics** that companies can use when developing products
  3. Provide **recommendations on improving ADLs**

Selected Current (First) Generation Protocols for Dynamic Service Discovery

3-Party Design

2-Party Design


WILEY

SLP for Enterprise Networks

Implementing and Deploying a Dynamic Resource Finder

Robert St. Pierre
James Kempf

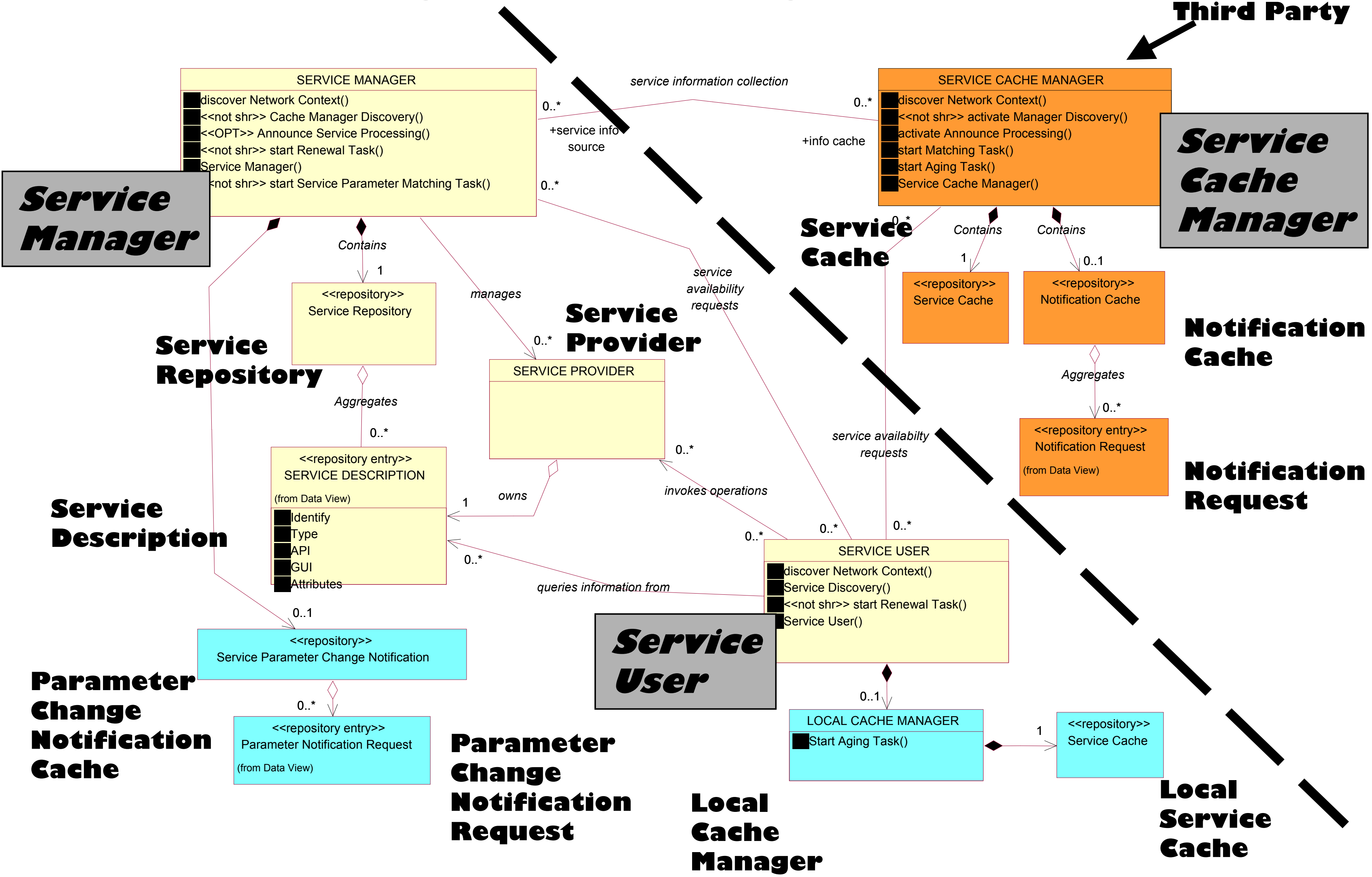
Adaptive 2/3-Party Design

Vertically Integrated 3-Party Design

Network-Dependent 3-Party Design

Network-Dependent 2-Party Design

Two Party vs. Three Party Architectures



Third Party

Service Manager

SERVICE MANAGER
- discover Network Context()
- <<not shr>> Cache Manager Discovery()
- <<OPT>> Announce Service Processing()
- <<not shr>> start Renewal Task()
- Service Manager()
- <<not shr>> start Service Parameter Matching Task()

Service Cache Manager

SERVICE CACHE MANAGER
- discover Network Context()
- <<not shr>> activate Manager Discovery()
- activate Announce Processing()
- start Matching Task()
- start Aging Task()
- Service Cache Manager()

service information collection
0..*
+service info source
0..*
+info cache
0..*

Service Cache

0..*
Contains
1
<<repository>>
Service Cache

Contains
0..1
<<repository>>
Notification Cache

Notification Cache

Aggregates
0..*
<<repository entry>>
Notification Request
(from Data View)

Notification Request

Contains
1
<<repository>>
Service Repository

Service Repository

manages
0..*

Service Provider

SERVICE PROVIDER

service availability requests

Aggregates
0..*
<<repository entry>>
SERVICE DESCRIPTION
(from Data View)
- Identify
- Type
- API
- GUI
- Attributes

Service Description

owns
1
0..*

service availabilty requests

invokes operations
0..*
0..*
0..*
0..*

queries information from

SERVICE USER
- discover Network Context()
- Service Discovery()
- <<not shr>> start Renewal Task()
- Service User()

Service User

0..1
<<repository>>
Service Parameter Change Notification

Parameter Change Notification Cache

0..*
<<repository entry>>
Parameter Notification Request
(from Data View)

Parameter Change Notification Request

0..1
LOCAL CACHE MANAGER
- Start Aging Task()

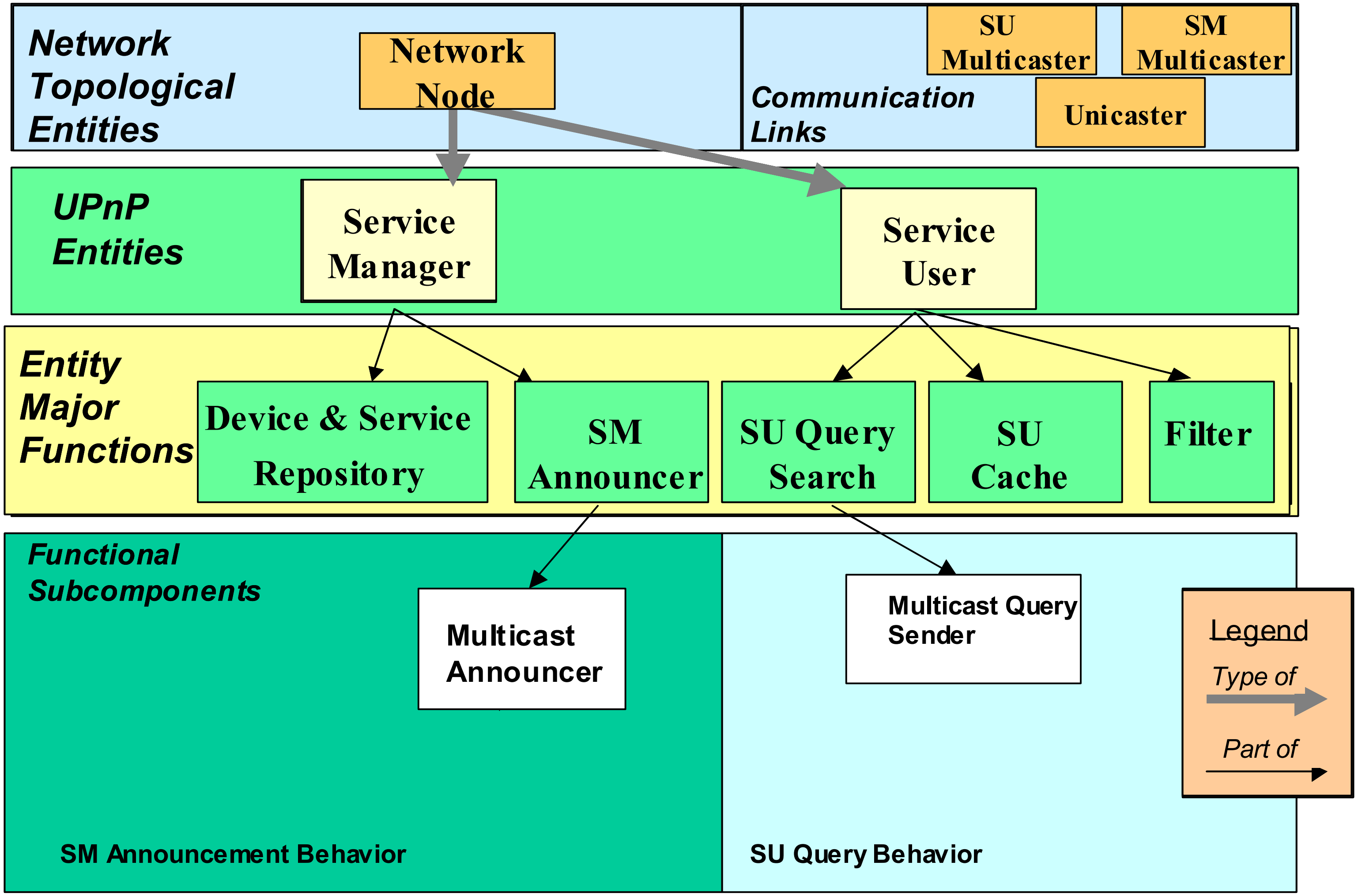
1
<<repository>>
Service Cache

Local Cache Manager

Local Service Cache

Layered View of Prototype UPnP Architecture in Rapide

Derived from SEI Architectural Layers Approach



How Do Service Discovery Architectures Propagate Changes During Link Failure?

- **Change Propagation:** In both two-party and three-party architectures changes in critical characteristics of Service Descriptions (SDs) must propagate from Service Managers (SMs) to Service Users (SUs) that already hold copies of the SDs.
  - Change propagation may take place through polling, eventing, or ad-hoc announcements – How do these strategies compare?
  - Does the existence of a third party (i.e., Service Cache Manager, or SCM) improve or hinder performance?
- **Approach:** develop a series of failure test scenarios and metrics for comparing and contrasting the alternative architectures with regard to
  - **Amount of inconsistent time** – sum of time that *$SM_k(SD_i)$ not equal to $SU_j(SD_i)$* for all i, j, k
  - **Change propagation latency** - time delay from [*$SM_k(SD_i)$ not equal to $SU_j(SD_i)$*] until [*$SM_k(SD_i)$ equal to $SU_j(SD_i)$*]
  - **Change propagation overhead** - number of messages in interval from [*$SM_k(SD_i)$ not equal to $SU_j(SD_i)$*] until [*$SM_k(SD_i)$ equal to $SU_j(SD_i)$*]

How Do Service Discovery Architectures Recover Consistency After Link and Node Failures?

- **Discovery and Recovery**: In both two-party and three-party architectures, SMs, SUs, and SCMs (where applicable) strive to maintain consistent descriptions (SDs) about discovered services and about event notifications. Link and node failures may lead to temporary loss of information about discovered services. Once failures are repaired, the information must be recovered.

- **We seek to develop metrics** to compare and contrast different service-discovery architectures and specifications. For example:
  - How do discovery latencies and overheads compare?
  - How do event registration latencies and overheads compare?
  - How do recovery latencies and overheads compare?

Can the Response of Service Discovery Processes Be Improved?

- Emerging designs for military fault-tolerant systems (e.g., OpenWings, OASIS, CoABS) rely on discovery-based component architectures to enable self-organizing and self-healing behavior
- The discovery protocols underlying such systems include mechanisms that permit network elements to continue to function as the topology varies
- However, many performance aspects of these protocols appear sensitive to parameter settings whose optimum values depend upon network topology
- Such **parameters** may be manually configured and **tuned in** relatively small, static environments, but their management in larger, **highly dynamic environments** cannot be performed manually

Investigating If Improvements Can Be Achieved Through Self-Adaptation

- **Model and analyze protocols** (UPnP, Jini, or SLP) **as specified**
  - develop SLX simulation models for each protocol
  - establish performance benchmarks based on default or recommended parameter values and on required or most likely implementation of behaviors
- **Investigate distributed adaptation algorithms** to control parameter values (and also consider selected adaptive behaviors)
  - devise several algorithms to adjust control parameters in each protocol
  - compare performance of each algorithm against benchmark performance
  - select most promising algorithms for further development
- **Implement and validate selected algorithms** in publicly available reference software
  - modify available implementation of UPnP, Jini, or SLP
  - deploy in service-discovery test bed (now under development at NIST)
  - validate simulated results with live experiments

Presentation Roadmap

- Project Objectives, Motivation, and Goals
- Modeling & Analysis
  - Architecture-based approach
  - Generic UML structural model
  - Specific models instantiated with Architecture Description Language
- Previous Work
  - Verifying our approach - using Jini as an example
- Overview of On-Going Work
  - How do different service discovery architectures respond to node and link failures?
  - How can these responses be improved?
- Plans for Future Work

Extending Generic UML Model to Encompass Message Exchanges and Assertions

- Can the messages exchanged among classes in our UML structural model be unified into a common vocabulary of message types and message attribute values?
- Can consistency conditions and other assertions be defined based on our unified structural and message-exchange models?
- Can consistency conditions be expressed to include temporal clauses that precisely bound the duration of any temporary inconsistencies permitted by a discovery protocol?
- Can these unifications be carried into our ADL model of specific discovery protocols, so that differences among the various architectures can be compared more directly?

Investigating Applicability of Architectural Models to Measure System Complexity

- Using the unified architectural models, create representations of various complexity metrics proposed in the literature, such as algorithmic information complexity,
-

To Delve More Deeply

Currently Available Paper

- Christopher Dabrowski and Kevin Mills, "Analyzing Properties and Behavior of Service Discovery Protocols using an Architecture-based Approach", accepted at DARPA-sponsored *Working Conference on Complex and Dynamic Systems Architecture*, to be held December 2001.

Available Software Artifacts

- Generic UML Structural Model (in Rational Rose format) of Discovery Protocols, including specific projections to Jini, UPnP, and SLP
- Rapide Models of Jini and UPnP (*in progress*).
- SLX Simulation Model of UPnP (*in progress*).

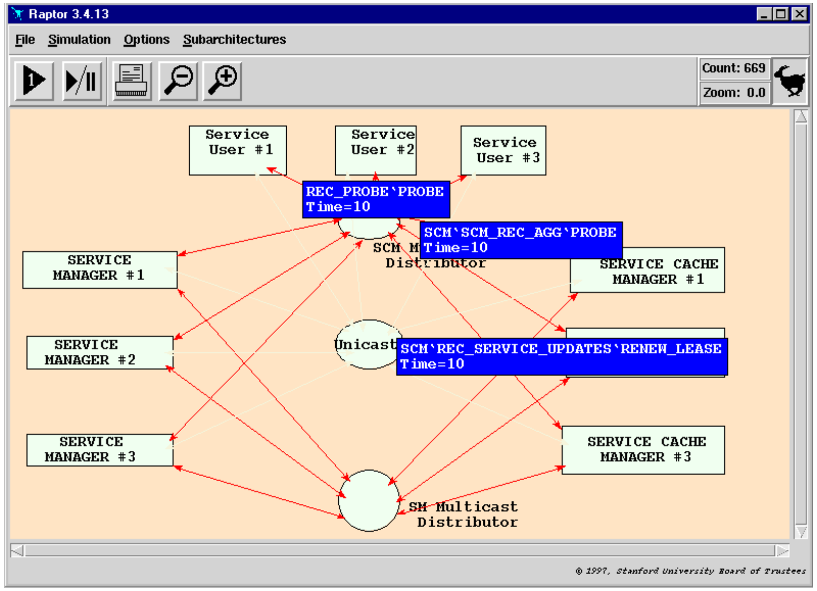
Related Web Sites

- http://www.itl.nist.gov/div897/ctg/adl/sdp_projectpage.html
- http://w3.antd.nist.gov/net_pc.shtml

Backup Slides

ITL Pervasive Computing Portfolio

Model & Analyze SDP Function, Structure, and Behavior



Objectives

(1) Provide increased understanding of the competing dynamic discovery services emerging in industry

(2) Develop metrics for comparative analysis of different approaches to dynamic discovery and assuring quality and correctness of discovery protocols

(3) Assess suitability of architecture description languages to model and analyze emerging dynamic discovery protocols

Technical Approach

- Develop ADL models from selected specifications for service discovery protocols and develop a suite of scenarios and topologies with which to exercise the ADL models
- Propose a set of consistency conditions & constraints that dynamic discovery protocols should satisfy
- Propose a set of metrics, based on partially ordered sets, with which to compare and contrast discovery protocols
- Analyze ADL models to assess consistency condition satisfaction, and to compare and contrast protocols

Recent Accomplishments:

- Developed a generic UML model encompassing the structure and function of Jini, UPnP, SLP, Bluetooth, and HAVi
- Projected specific UML models for Jini, UPnP, and SLP
- Completed a Rapide Model of Jini structure, function, and behavior
- Drafted and implemented a scenario language to drive the Rapide Jini Model.
- Developed a set of consistency conditions and constraints for Jini behavioral model; currently being tested using scenarios.
- Discovered significant architectural issue in interaction between Jini directed discovery and multicast discovery

Products & Contributions

- Rapide specifications of Jini, Universal Plug and Play (UPnP), and Service Location Protocol (SLP)
- Scenarios and topologies for evaluating discovery protocols
- Suggested consistency properties for service discovery protocols
- Suggested metrics, based on partially ordered sets (POSETs), for comparing and contrasting discovery protocols
- Paper identifying inconsistencies and ambiguities in service discovery protocols and describing how they were found
- Paper proposing consistency conditions for service discovery protocols, and evaluating how Jini, UPnP, and SLP fare
- Paper comparing and contrasting Jini, UPnP, and SLP at the level of POSET metrics